**20x20 Face Recognition using Hopfield Network Algorithm**

**Ivan (이반) – 2014758139**

First, we need to select some people’s face images as a sample for our training program. For this program, I’m using 3 20x20 pixel samples. The images will be blurred badly because 20x20 pixel is extremely small resolution.

C:\Users\Ivan\AppData\Local\Microsoft\Windows\INetCache\Content.Word\test1.jpg test1.jpg

C:\Users\Ivan\AppData\Local\Microsoft\Windows\INetCache\Content.Word\test2.jpg test2.jpg

C:\Users\Ivan\AppData\Local\Microsoft\Windows\INetCache\Content.Word\test3.jpg test3.jpg

**C++ - OpenCV Library**

I’m using C++ as a programming language in Microsoft Visual Studio 15. I also use OpenCV library to perform Image Processing. In this program, we need to read images, convert them to Grayscale, then we convert the Grayscale images to Black and White images using threshold method. Value greater than 128 will set to white, value less than 128 will set to black. After converting the image, we can get a binary values of each pixel; White = 0 , Black = 1.

You can find the code explanation by reading the comments section for each line of code.

**C++ Code**

/\*20x20 Face Recognition using Hopfield Network

Ivan - 2014758139\*/

#include<opencv2/core/core.hpp> //opencv library for image processing

#include<opencv2/highgui/highgui.hpp>

#include<opencv2/imgproc/imgproc.hpp>

#include <fstream>

#include<iostream>

#include<conio.h>

#include<stdlib.h>

#include<time.h> //to use srand (random value)

#include<math.h>

using namespace std;

int main() {

cv::Mat imgOriginal[3]; // input image

cv::Mat imgGrayscale[3]; // grayscale of input image

cv::Mat imgBinary[3]; // Binary image, black and white

imgOriginal[0] = cv::imread("test1.jpg"); // open image for training pattern

imgOriginal[1] = cv::imread("test2.jpg");

imgOriginal[2] = cv::imread("test3.jpg");

if (imgOriginal[0].empty() && imgOriginal[1].empty() && imgOriginal[2].empty()) { // if unable to open image

std::cout << "error: image not read from file\n\n"; // show error message on command line

\_getch();

return(0); // and exit program

}

cv::cvtColor(imgOriginal[0], imgGrayscale[0], CV\_BGR2GRAY); // convert to grayscale

cv::cvtColor(imgOriginal[1], imgGrayscale[1], CV\_BGR2GRAY);

cv::cvtColor(imgOriginal[2], imgGrayscale[2], CV\_BGR2GRAY);

imgBinary[0] = imgGrayscale[0] > 128; //convert to B&W using binary thresholding method

imgBinary[1] = imgGrayscale[1] > 128;

imgBinary[2] = imgGrayscale[2] > 128;

// declare windows

cv::namedWindow("ImageInput", CV\_WINDOW\_NORMAL); // CV\_WINDOW\_NORMAL which allows resizing the window

cv::namedWindow("ImageBW", CV\_WINDOW\_NORMAL); // or CV\_WINDOW\_AUTOSIZE for a fixed size window matching the resolution of the image

cv::namedWindow("ImageOutput", CV\_WINDOW\_NORMAL);

cv::moveWindow("ImageInput", 200, 350); //move window position

cv::moveWindow("ImageBW", 600, 350);

cv::moveWindow("ImageOutput", 1000, 350);

// Iterate through pixels.

int pixel[3][400]; //pixels total

int x;

for (int i = 0; i <3; i++) {

x = 0;

for (int r = 0; r < imgBinary[i].rows; r++) { //read pixel value from binary image row

for (int c = 0; c < imgBinary[i].cols; c++) { //read pixel value from binary image column

pixel[i][x] = imgBinary[i].at<char>(r, c) + 1; //save the binary value to pixel variable

x++;

}

}

}

cout << "Hopfield Network - Face Recognition (20x20):" << endl << endl;

srand(time(NULL)); // use current time to seed random number generator

int n = 400; // size of each pattern = number of neurons

int i, j, k, sum;

// Each row is a separate pattern to learn (n bits each).

cout << "Training patterns from Image:" << endl << endl;

int m = 3; //number of pattern

int\* pattern = new int[m\*n];

for (j = 0; j < m; j++) // rows

{

for (i = 0; i < n; i++) // columns

{

pattern[j\*n + i] = pixel[j][i]; //binary value from image's pixel saved to pattern variable

cout << pattern[j\*n + i]; //print the binary value of the image

}

cout << endl<<endl;

}

cout << endl;

// calculate the weight matrix (symmetric and square)

// w[i,j]=w[j,i] & i!=j (i==j => w=0)

int\* w = new int[n\*n];

for (j = 0; j<n; j++)

for (i = j; i<n; i++)

if (i == j)

w[j\*n + i] = 0;

else

{

sum = 0;

for (k = 0; k<m; k++)

sum += (pattern[k\*n + i] \* 2 - 1)\*(pattern[k\*n + j] \* 2 - 1); // w[i,j]=(2Xi-1)(2Xj-1) formula

w[j\*n + i] = sum;

w[i\*n + j] = sum;

}

// print the weight matrix

cout << "The weight matrix:" << endl << endl;

for (j = 0; j<n; j++)

{

for (i = 0; i<n; i++)

printf("%2d ", w[j\*n + i]);

cout << endl;

}

cout << endl;

cout << "Please select Pattern-recognition Image Test:(1 or 2 or 3) "<<endl;

int selectedPattern;

cin >> selectedPattern;

if (selectedPattern == 1) {

cv::imshow("ImageInput", imgOriginal[0]); // show Image1 in window

cv::imshow("ImageBW", imgBinary[0]); // show Black and White image in window

}

else if (selectedPattern == 2) {

cv::imshow("ImageInput", imgOriginal[1]); // show Image2 in window

cv::imshow("ImageBW", imgBinary[1]); // show Black and White image in window

}

else if (selectedPattern == 3) {

cv::imshow("ImageInput", imgOriginal[2]); // show Image3 in window

cv::imshow("ImageBW", imgBinary[2]); // show Black and White image in window

}

selectedPattern = selectedPattern - 1;

cout << "Test pattern selected from Image:" << endl;

for (i = 0; i<n; i++)

{

cout << pattern[selectedPattern\*n + i]; //move selected pattern node to new variable

}

cout << endl << endl;

int\* neuron = new int[n]; // current state of the network

int\* neuron\_prev = new int[n]; // prev state of the network

for (i = 0; i<n; i++)

{

neuron[i] = pattern[selectedPattern\*n + i];

neuron\_prev[i] = neuron[i]; // initially prev state=current

}

cout << endl << endl;

// if state of the network stays unchanged for ? steps

// that means the network is converged to an answer

// so then exit the loop and printout the last state

int ctr\_unchg = 0;

// loop counter to ensure a stop just in case

// if the network becomes cyclic or chaotic

int ctr = 0;

while (ctr\_unchg<2000 && ctr<10000) // max 10000 loops allowed

{

// updating the network

for (k = 0; k<n; k++) // update the whole network

{

// Serial-Random updating:

// Randomly select a neuron and update its value

j = rand() % n;

sum = 0;

for (i = 0; i<n; i++)

if (i != j)

sum += neuron[i] \* w[j\*n + i];

if (sum >= 0) //thresholding

neuron[j] = 1;

else

neuron[j] = 0;

}

// if state of the network unchanged

// then increase the unchanged counter

// else reset it

bool changed = false;

for (k = 0; k<n; k++)

if (neuron[k] != neuron\_prev[k])

{

changed = true;

break;

}

if (changed == false)

ctr\_unchg++;

else

ctr\_unchg = 0;

// update the previous network state

for (k = 0; k<n; k++)

neuron\_prev[k] = neuron[k];

ctr++;

}

// if the network is converged then

// printout the last state of the network

if (ctr\_unchg >= 100)

{

cout << "Converged Node of Face Recognition (Final Node Result):" << endl << endl;

for (i = 0; i<n; i++)

cout << neuron[i];

cout << endl << endl;

// calculate the convergence error percentage

int sumDif = 0; // total number of differences

for (i = 0; i<n; i++)

if (neuron[i] != pattern[selectedPattern\*n + i])

sumDif++;

cout << "Convergence error percentage:" << 100 \* sumDif / n << endl; //print the error percentage

cv::imshow("ImageOutput", imgOriginal[selectedPattern]); //show Image Recognition Result if network is converged

}

else

cout << "The network did not reach the convergence limit set!" << endl;

// garbage collection

delete[]pattern;

delete[]w;

delete[]neuron;

delete[]neuron\_prev;

cv::waitKey(0); // hold windows open until user presses a key

return(0);

}

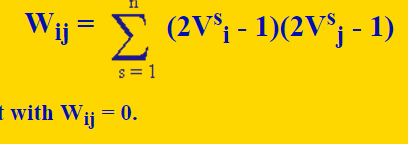
**Input Training Patterns**

These 3 input patterns are the binary values that have been derived from 3 black and white sample images. We use these values as the training pattern. Total node for 1 pattern is 20x20 = **400 nodes**.



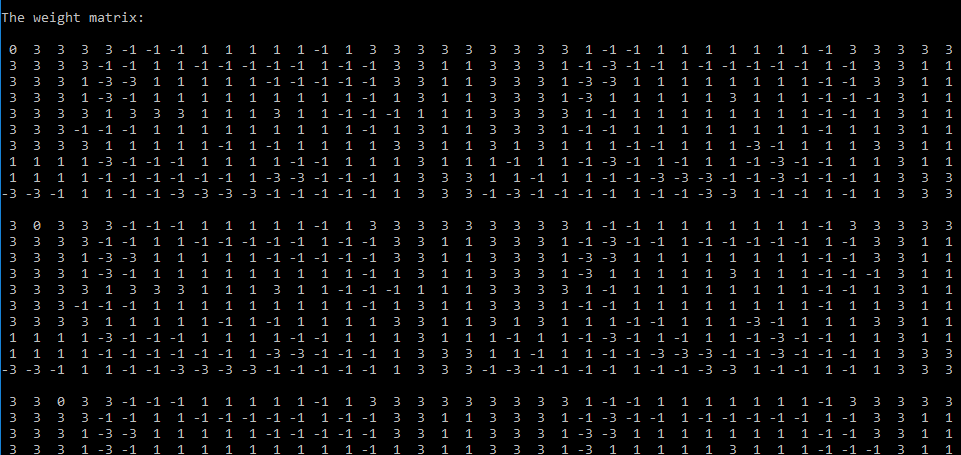
**Calculate Weight Values**

Using this formula



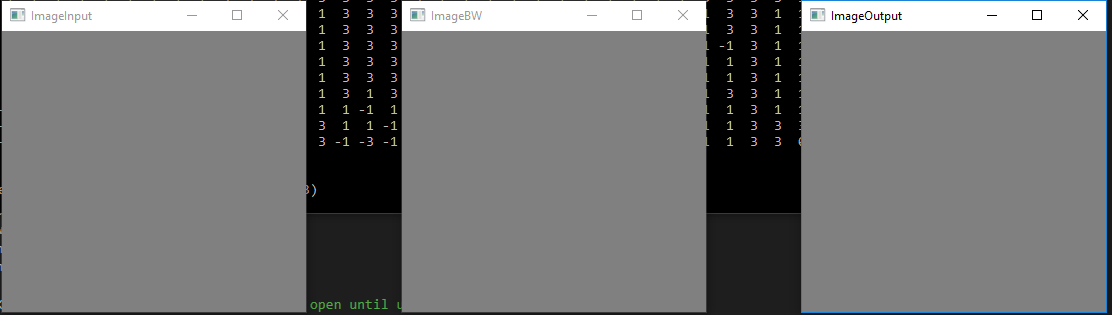
We create weight values for Hopfield network. In Hopfield, all nodes are connected to each other, which means each node is the input and the output for another node. That’s why the Weight is represented as Wij (node i and node j) where Wij = Wji.

Weight total **400^2 = 160000** weight values. I can’t show all 160000 values in this report because it’s too long.



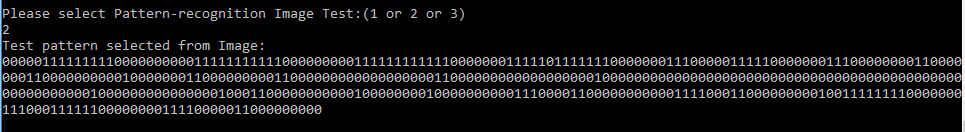
**Displaying Images**

Image Window still empty because the image that want to be recognized is not selected yet. Image Window consists 3 windows, for input, for conversion to black and white, and output image which corresponded to final node values of Hopfield Network training.



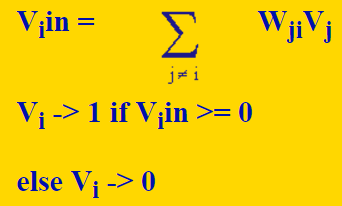
**Select Image that want to be Recognized**

I’m using 3 samples so we can only choose between those 3 images, and print the binary values. We can also input any values as long as it’s 400 nodes. For this project I’m using value from the image we’ve been used.



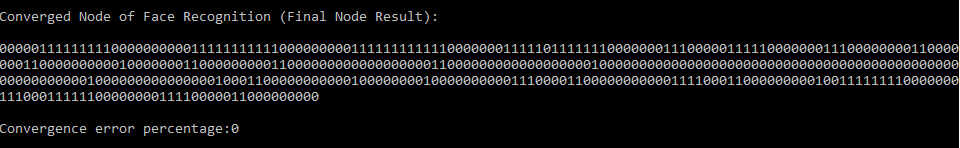
**Final Result after updating node**

Using this formula



Updating a node in a Hopfield network is very much like updating a perceptron. If you are updating node 3 of a Hopfield network, then you can think of that as the perceptron, and the values of all the other nodes as input values, and the weights from those nodes to node 3 as the weights. In other words, first you do a weighted sum of the inputs from the other nodes, then if that value is greater than or equal to 0, you output 1. Otherwise, you output 0.

I’m using Random updating method which means update the nodes in random order. This was the method described by Hopfield, in fact. You randomly select a neuron, and update it. Then you randomly select another neuron and update it. You keep doing this (Loop) until the system is in a stable state. When in stable state, none of the node has to be changed, the training program is stopped. Then print the converged node values (changed nodes).



**Displaying Images for Final Result**

Original Image that has been inputted – Conversion to Grayscale then to Black and White image – Successfully recognized face result

As we can see the ImageOutput window shows the same result as the ImageInput window, which mean the face of the person is successfully recognized.

